Machine Programming & the Future of Data-Driven Software Development

Justin Gottschlich

Principal AI Scientist
Director/Founder of Machine Programming Research
(Intel Labs)

Adjunct Assistant Professor
(University of Pennsylvania)

Contributors:
Todd Anderson, Saman Amarasinghe, Regina Barzilay, Michael Carbin, Alvin Cheung, Pradeep Dubey, Henry Gabb, Niranjan Hasabnis, Adam Herr, Jim Held, Tim Kraska, Insup Lee, Geoff Lowney, Shanto Mandal, Tim Mattson, Pranav Mehta, Abdullah Muzahid, Paul Petersen, Alex Ratner, Martin Rinard, Vivek Sarkar, Koushik Sen, Armando Solar-Lezama, Joe Tarango, Nesime Tatbul, Josh B. Tenenbaum, Jesmin Tithi, Javier Turek, Abdul Wasay, Rich Uhlig, Anand Venkat, Fangke Ye, Xin Zhang, Shengtian Zhou … and many others.

Overview

- Machine Programming Research @ Intel
- Discussion of The Three Pillars of MP
  - Separation of Intention is Critical
- The Bifurcated Space of Machine Programming
  - Stochastic and Deterministic
- Machine Programming Emphasis @ Intel
  - ControlFlag: a Self-Supervised Systems for MP

  (I blame Alex Ratner for ControlFlag ☺)

Definition: Machine Programming (MP) *is the automation of software and hardware development*

Machine Programming Research (MPR)

A New Pioneering Research Initiative at

Intel Labs' MPR Goals

Machine Programming (MP) is the automation of software and hardware development

Time:

Reduce development time of all aspects of software development

*Measured as 1000x+ improvement over human work performed today

Quality:

Better software than the best human programmers*

*Measured as superhuman correctness, performance, security, etc.

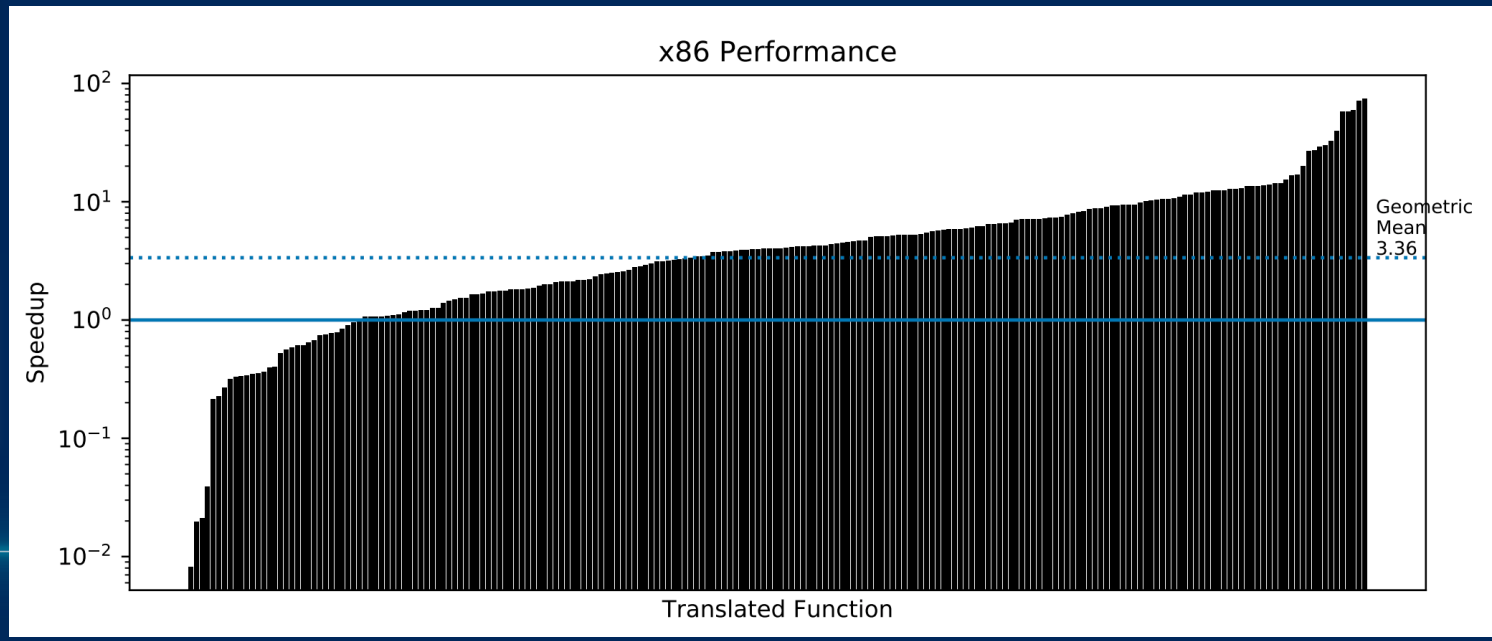
Intel Labs' MPR Goals

Machine Programm *d hardware development*

Concrete Data Point:

"Automatically Translating Image Processing Libraries to Halide" **(Ahmad et al., 2019)**

Time:

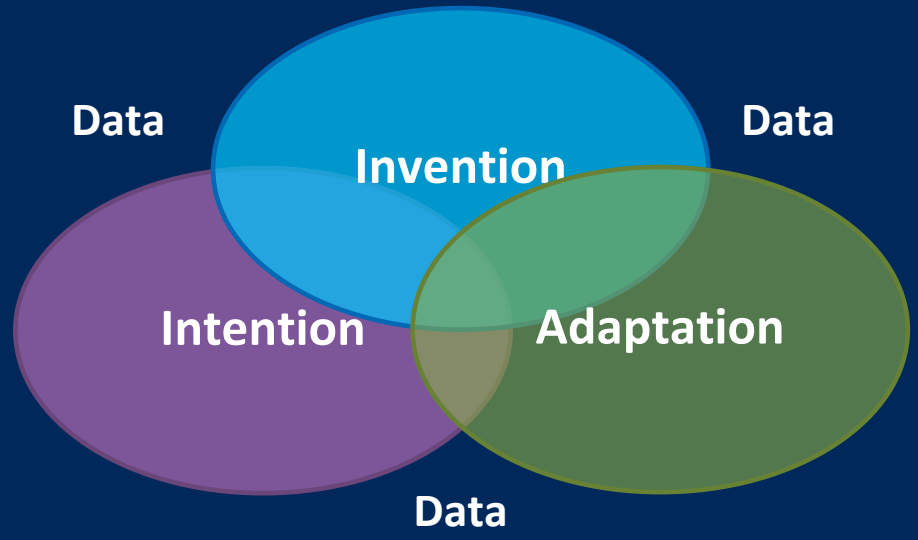


Quality:

The Three Pillars of Machine Programming

Machine Programming (MP) is the automation of software and hardware development

- **Intention:** Discover the intent of a programmer; lift meaning from software
- **Invention:** Create new algorithms and data structures; compositional novelty
- **Adaptation:** Evolve in a changing hardware/software world



The Three Pillars of Machine Programming

Justin Gottschlich
Intel Labs, USA
justin.gottschlich@intel.com

Armando Solar-Lezama
MIT, USA
asolar@csail.mit.edu

Nesime Tatbul
Intel Labs and MIT, USA
tatbul@csail.mit.edu

Michael Carbin
MIT, USA
mcarbin@csail.mit.edu

Martin Rinard
MIT, USA
rinard@csail.mit.edu

Regina Barzilay
MIT, USA
regina@csail.mit.edu

Saman Amarasinghe
MIT, USA
saman@csail.mit.edu

Joshua B. Tenenbaum
MIT, USA
jbt@mit.edu

Tim Mattson
Intel Labs, USA
timothy.g.mattson@intel.com

The Three Pillars of Machine Programming

Machine Programming (MP) is the automation of software and hardware development

- **Intention:** Discover the intent of a programmer; lift meaning from software
- **Invention:** Create new algorithms and data structures; compositional novelty
- **Adaptation:** Evolve in a changing hardware/software world

The Three Pillars of Machine Programming

Justin Gottschlich
Intel Labs, USA
justin.gottschlich@intel.com

Armando Solar-Lezama
MIT, USA
asolar@csail.mit.edu

Nesime Tatbul
Intel Labs and MIT, USA
tatbul@csail.mit.edu

Michael Carbin
MIT, USA
mcarbin@csail.mit.edu

Martin Rinard
MIT, USA
rinard@csail.mit.edu

Regina Barzilay
MIT, USA
regina@csail.mit.edu

Saman Amarasinghe
MIT, USA
saman@csail.mit.edu

Joshua B. Tenenbaum
MIT, USA
jbt@mit.edu

Tim Mattson
Intel Labs, USA
timothy.g.mattson@intel.com

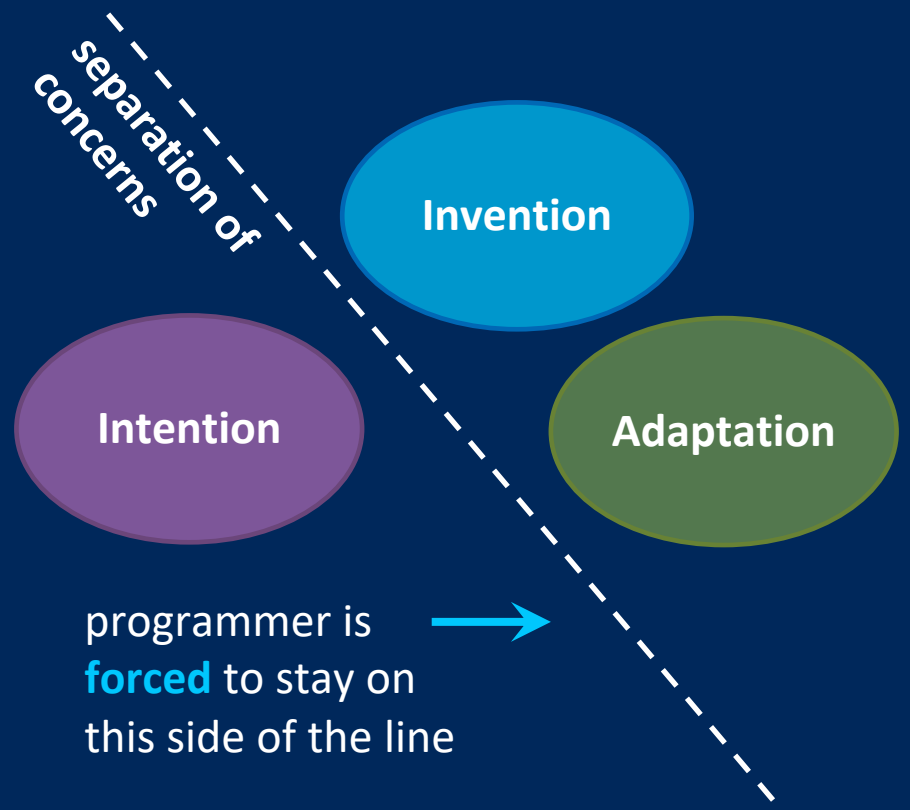
Separation of Intention is Critical

- Requires user only supply **core idea** (improving productivity)
- Enables machine to explore a **wider range** of possible solutions (improving MP-generated solutions)
- Enables automatic SW **adaptation & evolution**

We anticipate this separation will give rise to:

- **Intentional** Programming Languages

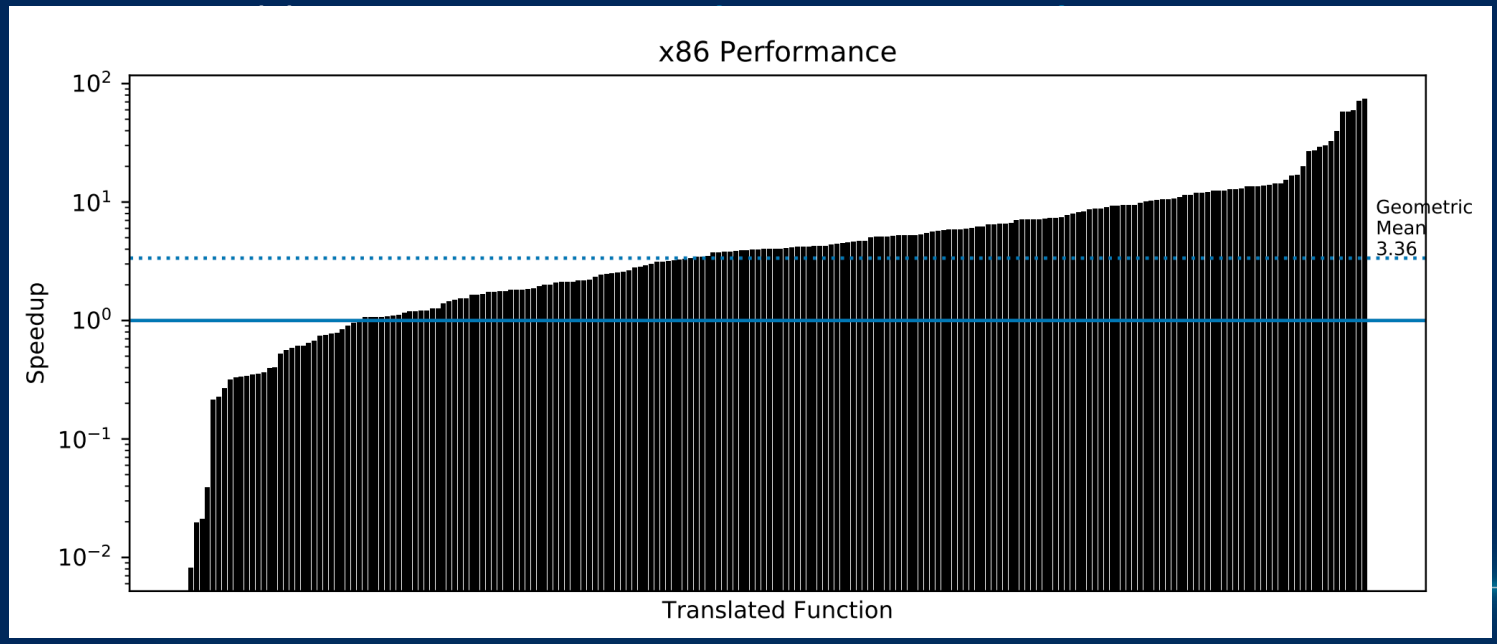
  Example: Halide/Verified Lifting (Adobe Photoshop)

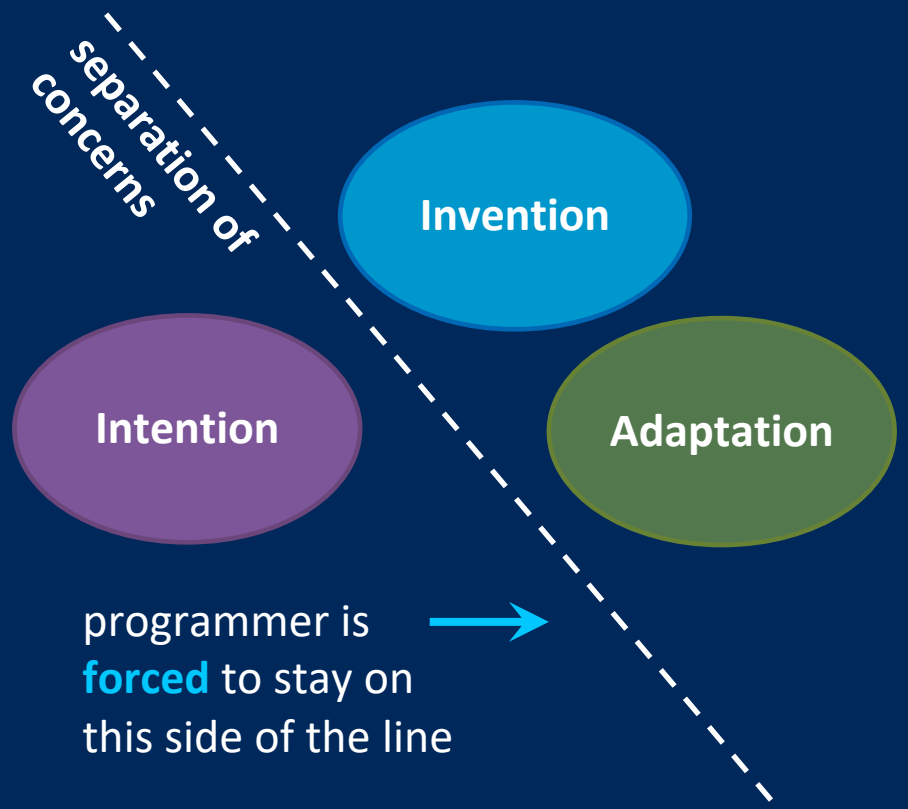


Separation of Intention is Critical

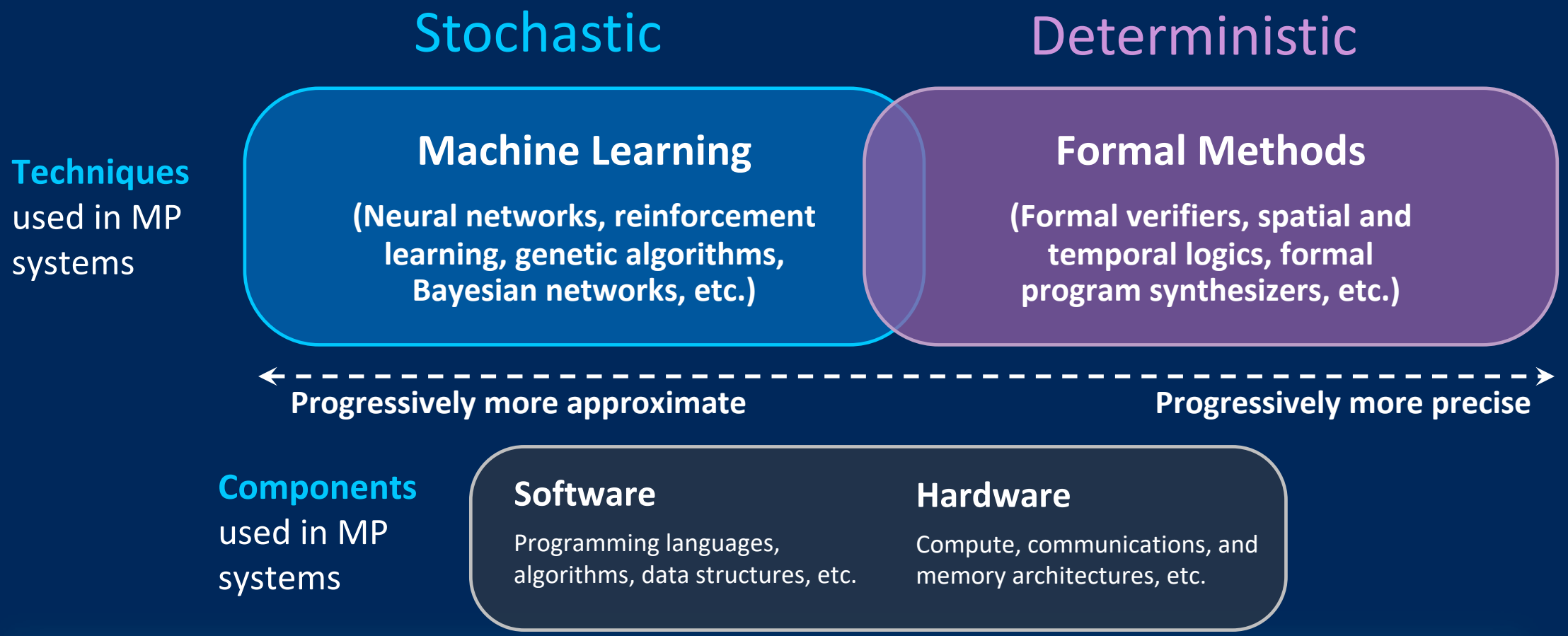
- **Leverages Separation of Intention from Invention & Adaptation**
- ***"Automatically Translating Image Processing Libraries to Halide"*** **(Ahmad et al., 2019)**

sible

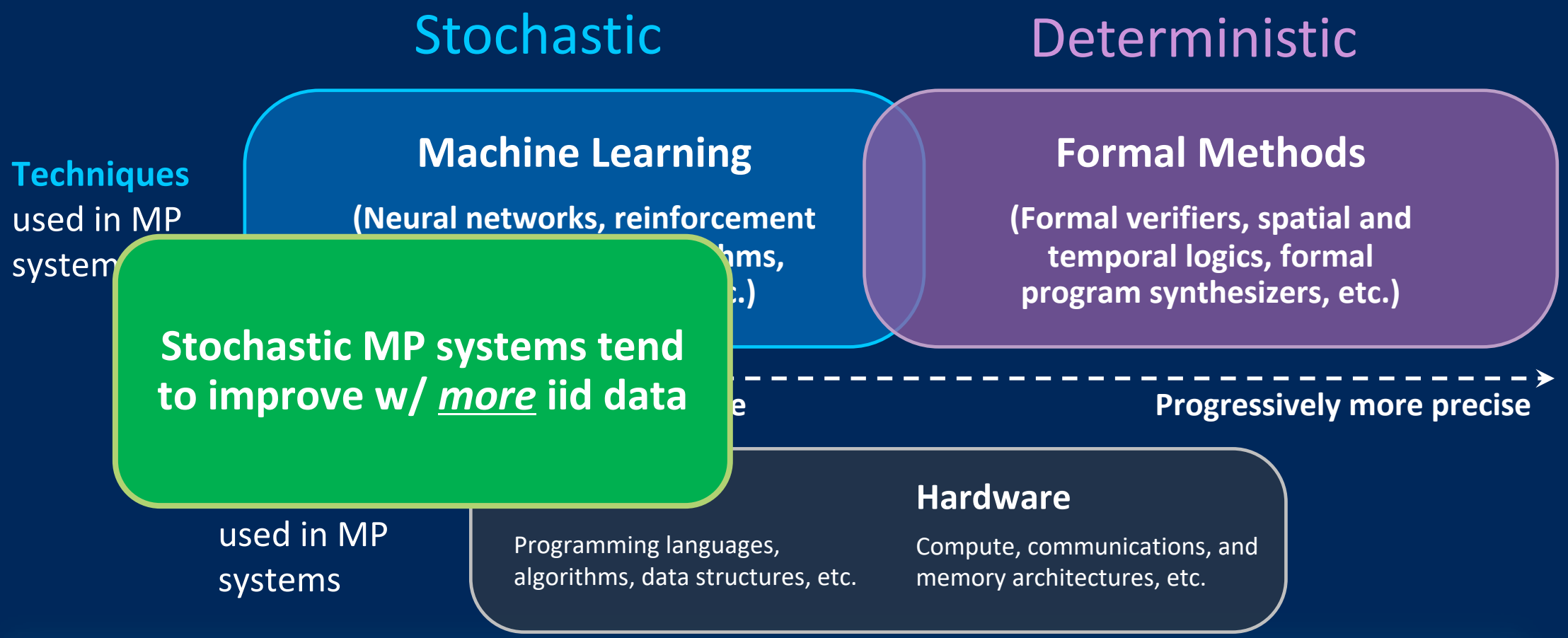




The Bifurcated Space of Machine Programming

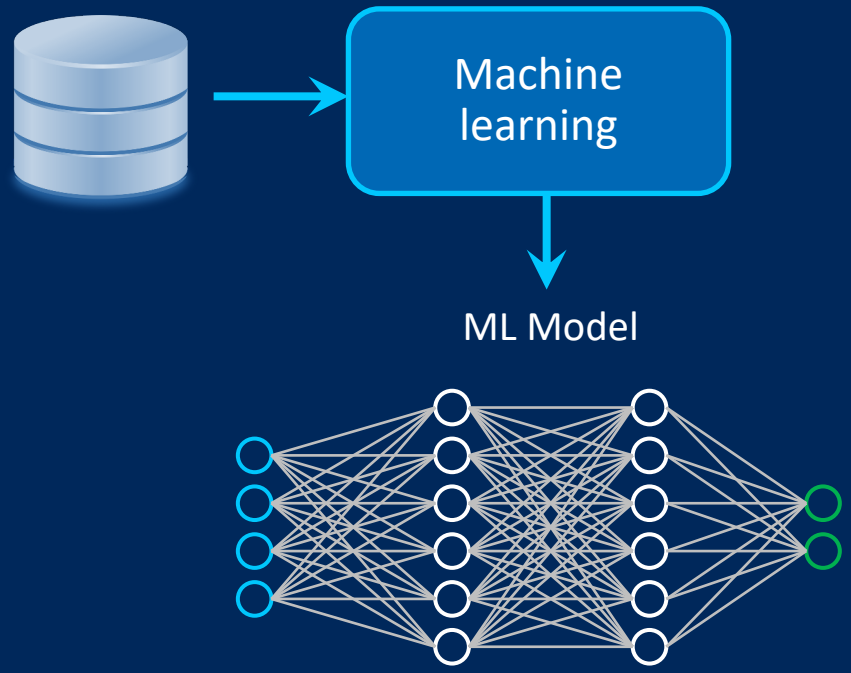


The Bifurcated Space of Machine Programming

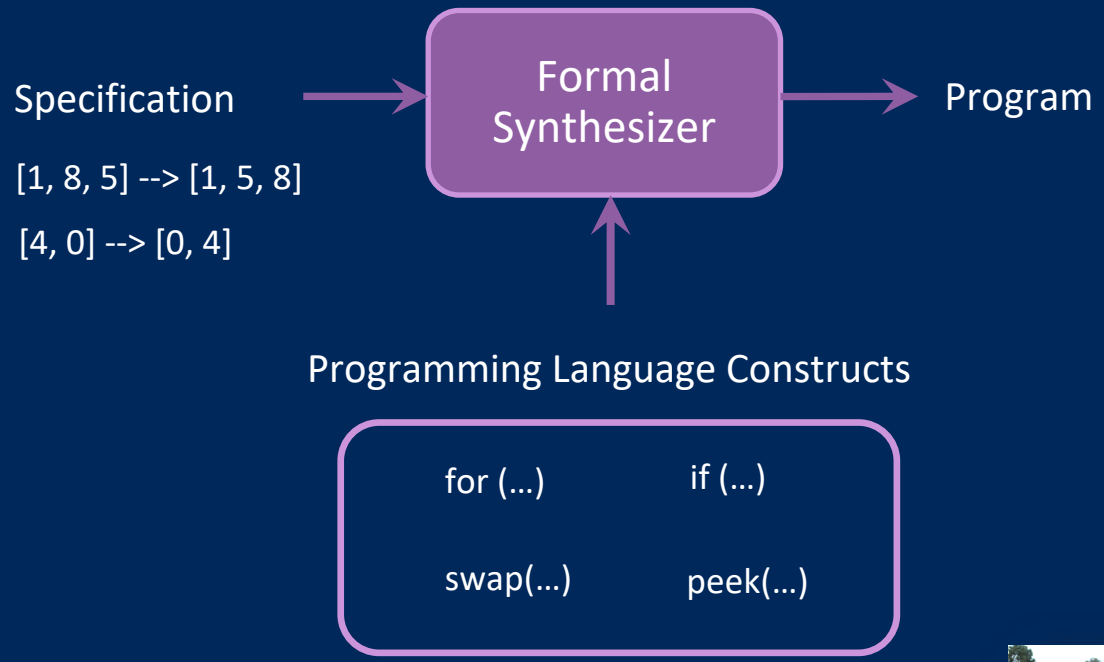


Concretizing The Two Sides of MP with Neuro-Symbolism

Stochastic (Neuro)



Deterministic (Symbolic)



Credit: Jeevana Inala & Armando Solar-Lezama

Numerous MP Efforts @ Intel

Debugging / Profiling / Productivity

- ControlFlag, MISIM, & AutoPerf

Automated Performance Extraction

- Inteon's Tiger Shark (Intel venture)
- MP-based general-purpose compiler (e.g., ML-learned code optimizations)

And Many More …

Numerous MP Efforts @ Intel

Debugging / Profiling / Productivity

- ControlFlag, MISIM, & AutoPerf

Beats SOTA by ~2x with 400k labeled data samples
Beats SOTA by ~5x with 1M labeled data samples
(independently confirmed by IBM/MIT)

Automated Performance Extraction

- Inteon's Tiger Shark (Intel venture)
- MP-based general-purpose compiler (e.g., ML-learned code optimizations)

And Many More …

Numerous MP Efforts @ Intel

Debugging / Profiling / Productivity

- ControlFlag, MISIM, & AutoPerf

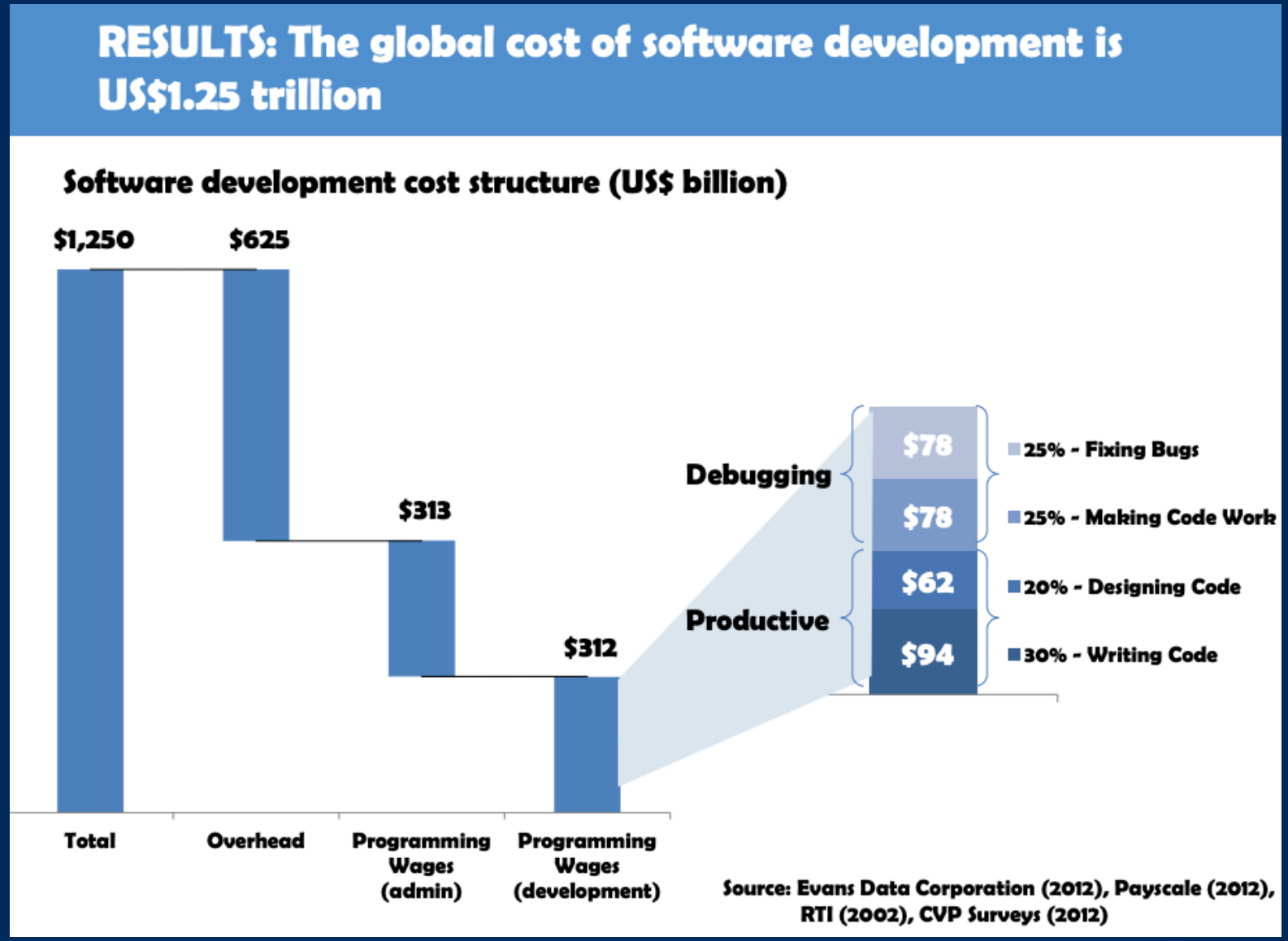
What can we build without labeled data?

Automated Performance Extraction

- Inteon's Tiger Shark (Intel venture)
- MP-based general-purpose compiler (e.g., ML-learned code optimizations)

And Many More …

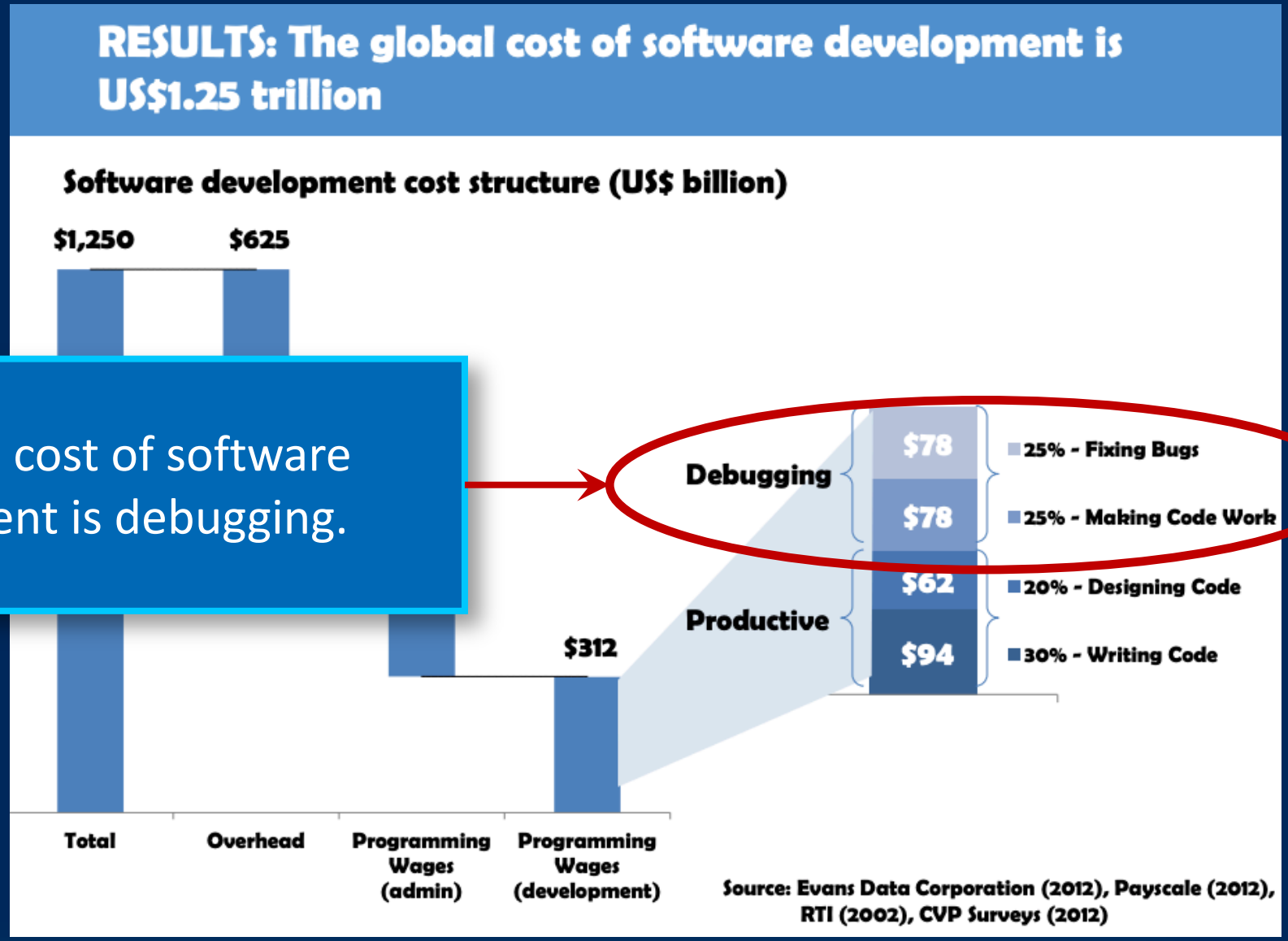
Productivity – Debugging



University of Cambridge (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.9611&rep=rep1&type=pdf)

Productivity – Debugging

50% of the cost of software development is debugging.



University of Cambridge (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.9611&rep=rep1&type=pdf)

Debugging: Finding Code Anomalies

What is a code anomaly?

- A piece of code that is irregular

Why care about code anomalies?

- Anomalous code can lead to defects, technical debt, delayed software development (hard to understand code), loss of customer trust

"*The Future of Data-Centric AI*" (Snorkel.ai event, Sept. 28, 2021)

Anomaly found in CURL (~30-year-old software)

CURL developers rewrite flagged piece of code found with ControlFlag

Re: Potential confusion in http_proxy.c and a recommendation

•*Contemporary messages sorted*: [by date] [by thread] [by subject] [by author] [by messages with attachments]
From: Daniel Stenberg via curl-library <curl-library_at_cool.haxx.se>
Date: Mon, 9 Nov 2020 23:51:20 +0100 (CET)

On Mon, 9 Nov 2020, Hasabnis, Niranjan via curl-library wrote:

> We believe that using "if (s->keepon > 1)" would eliminate this confusion
> and capture the intended semantics precisely.

I think you've pointed out code that could be written clearer, yes. But I think an even better improvement to this logic would be to use an enum or defined values that include all three used values as state names.

What do you think about my proposal over at:
https://github.com/curl/curl/pull/6193

https://curl.se/mail/lib-2020-11/0028.html

```
355 355            }
356     -          s->keepon = FALSE;
    356 +         s->keepon = KEEPON_DONE;
357 357            break;
358 358          }
359 359
360     -        if(s->keepon > TRUE) {
    360 +        if(s->keepon == KEEPON_IGNORE) {
361 361            /* This means we are currently ignoring a response-body */
362 362

▾ 6 lib/urldata.h
@@ -802,7 +802,11 @@ struct proxy_info {
802 802  /* struct for HTTP CONNECT state data */
803 803  struct http_connect_state {
804 804    struct dynbuf rcvbuf;
805     -  int keepon;
    805 +  enum keeponval {
    806 +    KEEPON_DONE,
    807 +    KEEPON_CONNECT,
    808 +    KEEPON_IGNORE
    809 +  } keepon;
806 810    curl_off_t cl; /* size of content to read and ignore */
807 811    enum {
808 812      TUNNEL_INIT,    /* init/default/no tunnel state */
```

Anomaly found in CURL (~30-year-old software)

CURL developers rewrite flagged piece of code found with ControlFlag

Re: Potential confusion in http_proxy.c and a recommendation

•*Contemporary messages sorted*: [by date] [by thread] [by subject] [by author] [by messages with attachments]
From: Daniel Stenberg via curl-library <curl-library_at_cool.haxx.se>
Date: Mon, 9 Nov 2020 23:51:20 +0100 (CET)

On Mon, 9 Nov 2020, Hasabnis, Niranjan via curl-library wrote:

> We believe that using "if (s->keepon > 1)" would eliminate this confusion
> and capture the intended semantics precisely.

I think you've pointed out code that could be written clearer, yes. But I think an even better improvement to this logic would be to use an enum or defined values that include all three used values as state names.

What do you think about my proposal over at:
https://github.com/curl/curl/pull/6193

https://curl.se/mail/lib-2020-11/0028.html

```
355 355            }
356     -          s->keepon = FALSE;
    356 +          s->keepon = KEEPON_DONE;
357 357            break;
358 358          }
359 359
360     -        if(s->keepon > TRUE) {
    360 +        if(s->keepon == KEEPON_IGNORE) {
361 361            /* This means we are currently ignoring a response-body */
```

6 lib/urldata.h

```
@@ -802,7 +802,11 @@ struct proxy_info {
802 802  /* struct for HTTP CONNECT state data */
803 803  struct http_connect_state {
804 804    struct dynbuf rcvbuf;
805     -  int keepon;
    805 +  enum keeponval {
    806 +    KEEPON_DONE,
    807 +    KEEPON_CONNECT,
    808 +    KEEPON_IGNORE
    809 +  } keepon;
806 810    curl_off_t cl; /* size of content to read and ignore */
807 811    enum {
808 812      TUNNEL_INIT,    /* init/default/no tunnel state */
```

Limitations in Existing Code Anomaly Detectors

Tools & techniques to identify software defects

- Testing (unit tests, QA, etc.)
- Static analysis
  - Compilers, linters

Limitations

- Continuous manual effort to maintain and update (i.e., adding new rules as things evolve)
- Manual efforts can be error-prone

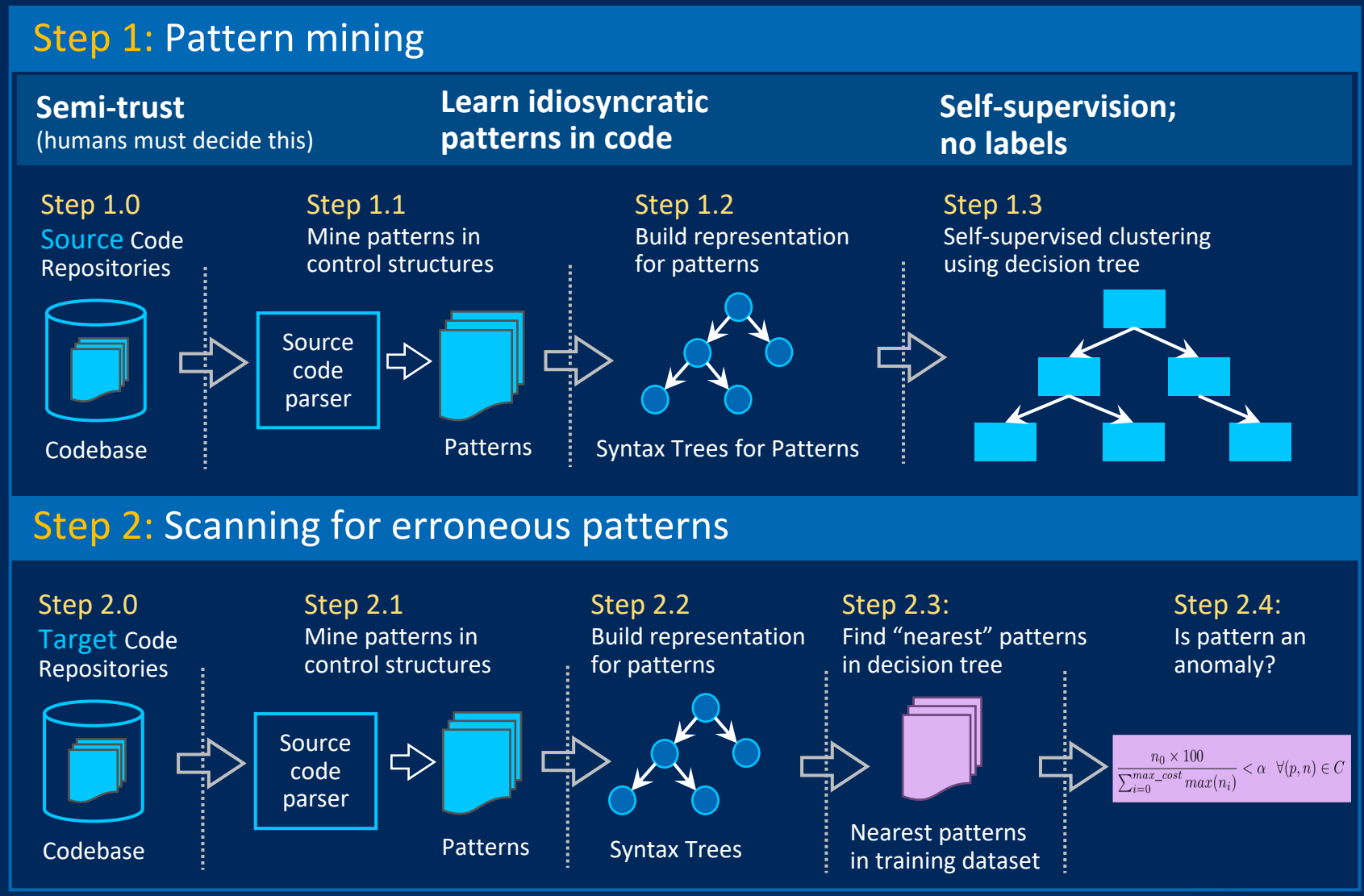
ControlFlag

A Self-Supervised Anomalous Code Detection System

Technical Lead:

Dr. Niranjan Hasabnis

Intel Labs



"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

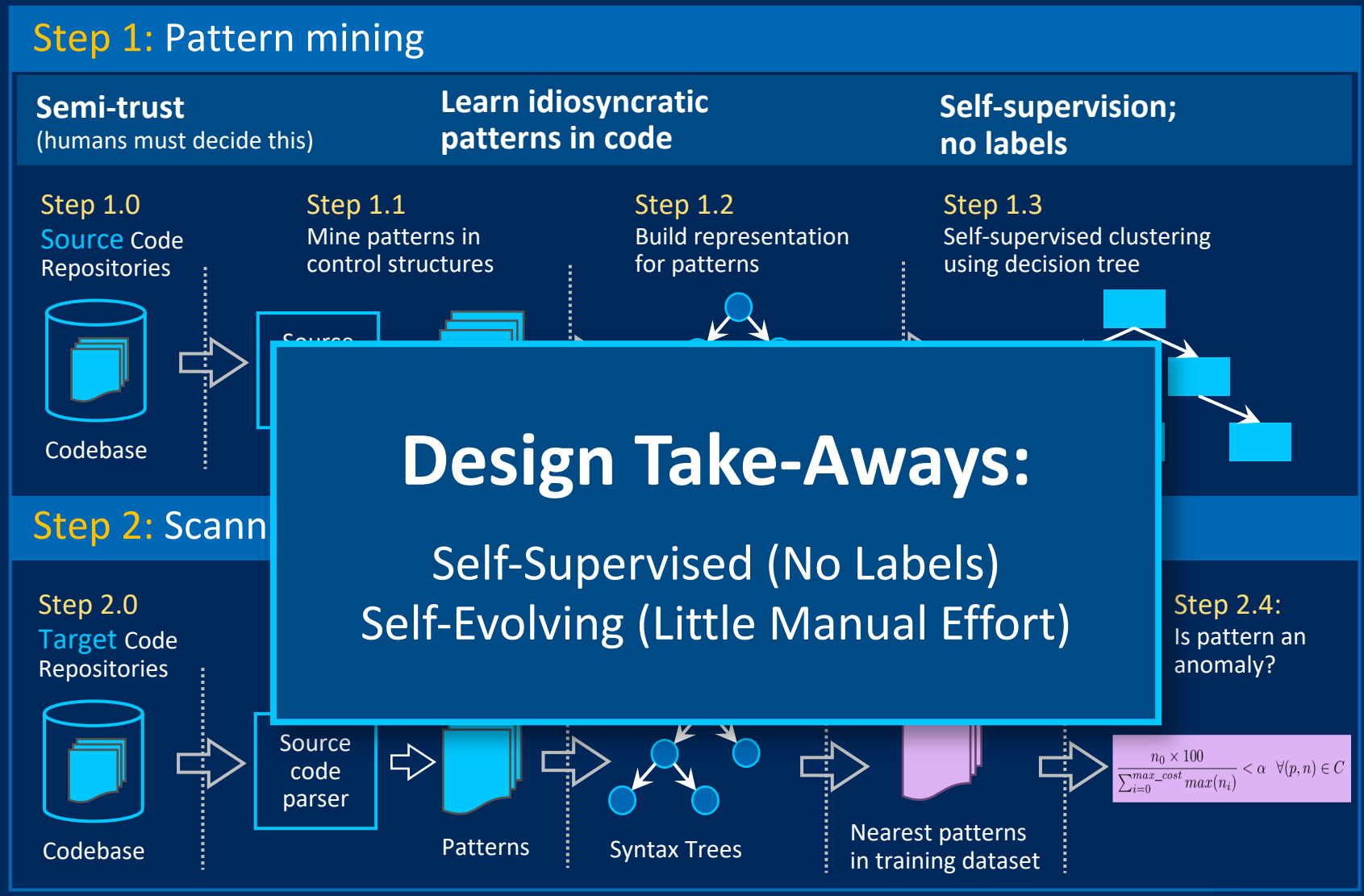
ControlFlag

A Self-Supervised Anomalous Code Detection System

Technical Lead:

Dr. Niranjan Hasabnis

Intel Labs



"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

Anomalies in Production-Quality, Open-Source Software

Evaluation: Setup

Training repository selection

- 6000 GitHub repos for C language having more than 100 stars
- 2.57M programs
- 1.1B Lines of code
- 38M patterns

Test repositories

- openssl, curl, ffmpeg
- git, vlc, lcx, lz4, reactos

Repo	GitHub stars	Found Anomalies	Scanned Expressions	Types of anomalies found
IoLanguage/io	2.3K	**5**	1635	Confusing expressions; missing parenthesis
Git/git	38.9K	**6**	6341	Confusing expression; character comparison using greater than or less than
Rubinius/rubinius	3K	**2**	10135	Character comparison using greater than or less than; missing parenthesis
FreeRADIUS/ freeradius-server	1.5K	**3**	20621	Character comparison using greater than or less than
Davidfstr/rdiscount	755	**4**	472	Character comparison using greater than or less than; missing parenthesis
Libharu/libharu	1.2K	**1**	2785	Character comparison using greater than or less than
Macournoyer/tinyrb	454	**3**	4369	Character comparison using greater than or less than
Rhomobile/rhodes	1K	**14**	76128	Confusing expressions; missing parenthesis; character comparison using greater or less than

"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

Anomaly Found in Proprietary & Deployed Software

Anomaly flagged by ControlFlag: in 12
if (address % 4)

```
1. void func() {
2.  uint32_t* p32;
3.  uint16_t* p16;
4.  uint32_t index = 0, other_index = <function_call>;

5.  for(j = 0; j < ..; j++) {
6.   if (array[j+1] == some_value) {
7.      index = j + 1;
8.      break;
9.   }
10. }

11. p16 = &array1[other_index].array2[index];

12. if ((uint32_t) &array1[other_index].array2[index] % 4) {
13.   p32 = (uint32_t*)(p16 - 1);
14.   *p32 = (*p32 & 0x0000FFFF) | (uin32_t) (mask);
15. } else {
16.   p32 = (uint32_t*) p16;
17.   *p32 = (*p32 & 0xFFFF0000) | some_other_mask;
18. }
19. }
```

An Example of ControlFlag's Finding

Three defects:

1. Duplicate expression in lines 11 and 12
2. Possible out-of-bounds memory access (memory error) in line 14
3. Information leak, security vulnerability in line 11

"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

RESULTS: Summary of 1st Proprietary Repo Analysis

Identified 104 potential defects

- 812 scanned files (.C and .H)
- 353K scanned lines of code
- 4600 scanned expressions

3 hours total analysis time (approx.)

- 56 Intel CPU cores

Description	Count	Comments
Anomalies that are critical bugs	**2**	Type error; memory error; security vulnerability
Anomalies that can lead to unwanted side-effects	**39**	Missing NULL check; possible divide by 0; missing return value check
Anomalies that point to confusing programming style	4	Double parenthesis around expressions, when not required
Anomalies that point to improvements in programming styles	59	Not using named constants; constant on right hand of equality;
Total unique anomalies reported	104	Not including false positives

"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

RESULTS: Summary of 2nd Proprietary Repo Analysis

Identified 191 potential defects

- 19K scanned files (.C and .H)
- 10.9M scanned lines of code
- 18.7K scanned expressions

8 hours total analysis time (approx.)

- 12 Intel CPU cores

Description	Count	Comments
Bugs found (confirmed by group)	5	Bitwise operation instead of Boolean logic operation
Confusing programming styles that could lead to bugs	22	Overly complex code E.g., ((xxxx[pstate].yyy & 0x1) >> 0)
Syntactic improvements to code according to standard style guides	164	Stylistic deviations from standards
Total unique anomalies reported	191	Not including false positives

"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

RESULTS: Summary of 2nd Proprietary Repo Analysis

Identified 191 potential defects

- 19K scanned f
- 10.9M scanned
- 18.7K scanned

8 hours total anal

- 12 Intel CPU core

Description	Count	Comments
Bugs found (confirmed by group)	5	Bitwise operation instead of Boolean logic operation
		plex code [pstate].yyy & 0x1) >> 0)
		viations from standards
		ng false positives

Working on a larger scan of ~65M lines of code, which identified 25,000 anomalies.

Number of files (.C and .H)	126,896
Number of expressions	1,374,028
Number of lines of code	64,690,054

Intel's partner is working to integrate ControlFlag as a permanent component of their continuous integration process.

"*ControlFlag: A Self-Supervised Idiosyncratic Pattern Detection System for Software Control Structures*" by Hasabnis & Gottschlich, MAPS '21

Conclusion

- Machine Programming Research charter
- Discussion of The Three Pillars of MP
  - Separation of intention, lifting code semantics
  - Intentional programming languages
- The Bifurcated Space in MP
  - Stochastic and Deterministic
- ControlFlag: A Self-Supervised Systems for MP
  - Demonstrable evidence to find latent defects in production-quality, deployed software
  - Early evidence this is a viable approach for MP

(I blame Alex Ratner for this observation ☺)

Future and Open Invitation for Collaboration

Future directions

- Growing MP investment across all of Intel, weak supervision likely to play critical role (where is Snorkel for MP, Alex?!)
- MPR is hiring PhD+ researchers; please reach out to me

Industrial and academic collaborations

- Teaching MP fundamentals at Berkeley and MIT, Fall 2021
- Numerous funded academic efforts in place and upcoming

Stay current with MP and our open-sourcing

- Intel's Website, LinkedIn, Twitter, and YouTube MP Channel
- Intel Innovation event, Oct. 27-28 (MP sessions and demos)

intel labs

Penn
University of Pennsylvania

Thank you!